BETA BASIC

Beta Basic aggiunge 26 tasti chiave nuovi e 10 funzioni a quelle già presenti sullo SPECTRUM. Inoltre , alcuni comandi esistenti sono stati migliorati e sono state aggiunte varie prestazioni come il cursore di linea lampeggiante e un Break per L/M. Vi sono 2 versioni del Beta Basic una per il 16K e un'altra per il 48K(la versione per il 16K gira sul 48K ma sottrae molta memoria).

Per caricarlo dovete dare il LOAD "bb48" oppure LOAD "bb16" a seconda della versione SPECTRUM che possedete. Verranno caricate 2 linee di BASIC la 0 e la 1. La linea 0 definisce le nuove funzioni. La linea 1 abbassa la RAMTOP(27168 per il 16K,59936 per il 48K) per far spazio alla parte principale del programma che occupa circa 5,5K e quindi la carica. Il caricamento è finito nel momento in cui compare il copyright del BetaBasic in fondo allo schermo. La linea 1 si cancellarà da sola non appena visualizzato il copyright e rimarrà solamente la linea 0. Un altro segno che il BetaBasic è residente e funzionante è il il cursore di linea lampeggiante. Il click della tastiera è stato allungato nella sua durata, se non vi piace date POKE 23609,0 per eliminarlo. Per caricare adesso un altro programma non date LOAD"" bensì MERGE"" altrimenti andrà persa la linea 0 e con essa le 10 nuove funzioni, mentre le nuove parole chiave funzionerebbero lo stesso. Se invece delle parole chiave volete usare gli UDG dovete prima dare il comando KEYWORDS 0. Potrete vedere inoltre un'aumentata velocità nell'esecuzione dei GOTO e GOSUB, anche i RETURN saranno più veloci e un return alla ultima linea sarà tanto veloce quanto un RETURN alla prima.Il NEW è adesso meno drastico e cancella tutto il programma in memoria tranne la linea 0 ed effettua un CLEAR. Una volta che avete scritto un programma potete salvarlo su nastro come normale. Quando dovete ricaricarlo prima di tutto caricate il BetaBasic se non è già presente in memoria e quindi caricate il vostro programma in memoria(una copia della linea 0 sarà stata salvata con il programma e quindi non è necessaria che usiata MERGE). Se scordate di caricare prima il BetaBasic i nuovi comandi compariranno come singoli caratteri che una volta dato il RUN provocheranno la comparsa del messaggio "Nonsense in Basic". Per ovviare a ciò potreste usare:

MERGE"bb48":GOTO 1

e l'equivalente con"bb16". L'uso di GOTO 1 è reso necessario dal fatto che il caricamento con MERGE impedisce l'autorun.

Tutti i nuovi comandi sono ottenuti con un tasto solo(nel classico modo Sinclair) ma bisogna prima entrare nel modo grafico, tutti tranne EDIT. Una volta nel modo grafico quasi tutti i tasti daranno un comando. Dopo aver premuto un tasto si esce automaticamente dal modo grafico. La localizzazione dei nuovi comandi sulla tastiera e descritta nel diagramma della tastiera sulle istruzioni e ripetuta vicino alla spiegazione di ciascuno di essi. La sintassi per ciascun comando e spiegato nel capitolo esplicativo di ciascuno di essi, tutto ciò che nelle intestazioni viene compreso tra due segni "%" significa che può essere omesso. Il controllo della sintassi avviene all'ingresso di ogni nuova linea o comando e può provocare la comparsa di messaggi d'errore del

Sinclair e del Betabasic stesso a secondo dei casi. Negli esempi dati nelle pagine che seguono si è cercato di evitare di fornirepiù di uno al massimo 2 nuovi comandi nellà stesso esempio. Ciò significa che non è sempre state fornito l'esempio più elegante.

ALTER % descrizione dell'attributo % TO descrizione dell'attibuto
Taste: A

ALTER permette un'elevata capacità manipolativa degli attributes dello schermo (che contengono le informazioni INK,PAPER,BRIGHT,FLASH, per la posizione di ciascun carattere.) Nella sua forma più semplice ALTER può cambiare l'INK e il PAPER senza cancellare lo schermo.

```
100 PRINT AT 10,10;"TEST":PAUSE 50: ALTER TO PAPER 1
```

cambierà il colore dello sfondo di ciascuna posizione di carattere in blue, potete inoltre cambiare il colore dell'intero schermo in qualsiasi altra combinazione:

```
ALTER TO PAPER2,INK7, FLASH 1
```

E' possibilà un minimo si selettività riguarde a quali posizioni di carattere verranno interessate, indicando gli attributes di queste posizioni da modificare prima diTO.

```
ALTER INK 7 TO INK 0
```

cambierà il colore di stampa da bianco in nero. E' possibile creare immagini, anche complesse, che compaiano improvvisamente, basta prima creare l'immagine con la stampa dello stesso colore dello sfondo e quindi con ALTER cambiare INK o PAPER a tutto lo schermo. Si possono fare statements complessi come:

```
ALTER INK3,BRIGHT 1,PAPER 7 TO INK 5, FLASH 1
```

che agira solo sulle posizioni di carattere con INK 3, BRIGHT 1,PAPER7.

Il seguente programma mostra alcune delle possibilità di questo comando, crea una grande lavagna lampeggiante. Cercate di cambiare l'effetto cambiando i colori di ALTER. La frequenza di lampeggiamento può essere alterata cambiando le linee 140 E 190.

(vedi programma pag.3 istruzioni in Inglese)

AUTO %numero di linea% %incremento%
Taste :6

AUTO attiva la numerazione automatica delle linee e il computer creerà per voi il numero di linea. Se omettete il valore dell'incremento verrà assunto come pari a 10. Se AUTO viene inserito senza alcun valore la prima linea creata sarà quella indicata dal cursore di linea lampeggiante più 10. Per disattivarlo usate AUTO 0 oppure introducete un numero di linea oltre 9984, oppure scrivete qualcosa, dovete sempreperò cancellare il numero di linea proposte e scrivere il vostro messaggio. Dopo che è stato dato l'ENTER a una linea la successiva ha il valore della precedente più il valore d'incremento

AUTO	- dalla attuale linea + 10, Incremento 10
AUTO 100	- dalla linea 100, incremento 10
AUTO 100,5	- dalla linea 100, incremento 5

BREAK
Tasto: SHIFT-SPACE (senza entrare in modo grafico)

(Break non è una parola chiave). Il normale Break del Sinclair BASIC è perfettamente adeguato per la maggior parte degli usi, purtuttavia colore che sono interessati al linguaggio macchina saranno spesso capitati nella necessità di spegnere la macchina perchè il codice macchina da loro introdotto era entrato in un loop infinito da cui non è possibile uscire se non con questo sistema. Con il BetaBasic invece una volta che abbiate usato uno dei comandi CLOCK, viene attivato un particolare sistema di break che funziona anche se la routine non funziona. Se premete i tasti SHIFT-SPACE per più di un secondo si attiva questo sistema di break che può essere molto utile anche nel Basic se avete disattivato l'azione di "Step in input" o di "Break into program" tramite l'uso non molto accorto di ON ERROR.

Nota:
1) Se siete tornati in qualche modo al messaggio copyright della Sinclair significa che il sistema di Break è stato disabilitato.
2) (per i programmatori in L/M) Se volete mantenere la funzione del nuovo Break non disabilitate gli interrupt.

CLOCK numero o stringa
tasto: 6

CLOCK controlla un orologio 24hh giudato dall'interrupt, che può visualizzare l'ora esatta nell'angolo in alto a destra dello schermo in ore, minuti e secondi. Quando viene raggiunto un orario da voi fissato, ammesso che l'abbiate fatto, viene generato un suono e/o viene eseguita una subroutine sempre da voi preparata. "Guidata dall'interrupt" singifica che l'orologio cammina vengono eseguiti i programmi o voi state programmando. Solo durante le operazioni con la cassetta e durante i BEEP l'orologio si fermerà. Per controllare quali delle capacità di CLOCK debbano entrare in azione bisogna che CLOCK sia seguito da un espressione numerica il cui valore sia tra 0 e 7. I risultati saranno i seguenti:

MODO	ALLARME GOSUB	ALLARME SONORO	VISUALIZZAZIONE
0	NO	OFF	OFF
1	NO	SPENTO	ACCESO
2	NO	ACCESO	SPENTO
3	NO	ACCESO	ACCESO
4	SI	SPENTO	SPENTO
5	SI	SPENTO	ACCESO
6	SI	ACCESO	SPENTO
7	SI	ACCESO	ACCESO

Leorelogio partirà non appena venga selezionato uno dei modi qualsiasi qui sopra elencati. La partenza, se non è avvenuta regolazione, avverrà da un orario 00:00:00. Se volete vederlo date CLOCK 1.
Per regolare l'ora bisogna introduràa sotto forma di stringa:

CLOCK"9:29:55"

per esempio. In effetti i due punti sono sencessari in quanto il comante clock ignorerà tutto fuorhcèi primi 6 numeri della stringa

(ma con una eccezione). Se una stringa contiene meno di 6 numeri gli altri verranno assunti come zero. Ad esempio:

CLOCK"mky10"

regolerà l'orologio per le ore 10000:00.

L'eccezione menzionata prima riguarda la "a" (non ha importanza se maiuscola o minuscola). Questa lettera fa considerare tutti i numeri che vengono dopo come l'ora a cui deve essere regolato il sitema di allarme:

CLOCK"a 6:20"

regolerà l'allarme alle 6 e 20 che attiverà o l'allarme sonoro o la subroutine a seconda della scelta effettutata(oppure entrambe o nessuno) Se l'allarme subroutine è attivato il computer SOLO SE STA ESEGUENDO UN PROGRAMMA salterà all'esecuzione della subroutine. Il programma potrebbe essere semplicissimo come:

10 GOTO 10

oppure complicato come un Word Processing o un gioco. Una volta che l'ora sia raggiunta CLOCK aspetta che lo statemente in esecuzione finisca di essere eseguito,il che richiederà un po' di tempo se lo statement è un PAUSE o un INPUT, dopo di che passa all'esecuzione della subroutine. Questo GOSUB non verrà eseguito da un programma che sia interamente in L/M. La linea a cui il computer dovrà saltare a quella data ora viene indicata con CLOCK (numero di linea) dove il numero di linea deve essere compreso tra 8 e 9999. Nella subroutine che verrà usata i nomi delle variabili devono essere differenti da quelli del programma principale a meno che non intendiate alterarli intenzionalmente. Se desiderate che la subroutine salvi dei dati e il programma principale effettua un CLEAR e un RUN dovrete spostare i dati in zone di memori particolari altrimenti verranno persi.

Applicazioni possibili per la subroutine includono variazioni sul nurmale allarme. Es.: il suonare una qualche melodia e la contemporanea esecuzione di un disegno sullo schermo, segnalazione dell'ora(chime) (l'ora corrente è disponibile:come funziene e può essere usata nella subroutine, guardate FNT$). Appassionati di elettronica possono raccogliere dati dall'esterno ogni ora o minuto e in ogni caso quandodesiderano in questo modo:

```
8999 STOP
9000 PRINT "Subroutine attivata"
9010 LET pointer = PEEK 27000: POKE pointer +27001, IN 127
9020 LET pointer = pointer +1: IF pointer >100 THEN LET pointer=0
9030 POKE 27000,pointer: LET Z$= FN T$()
9040 LET hours= VAL Z$ (1 TO 2): LET mins=VAL Z$ (4 TO 5)
9050 LET mins=mins+1: IF mins=60 THEN hours=hours+1:LETmins=0
```

Non date il RUN ? invece date i seguenti comandi

```
CLEAR 26999:POKE27000,0:CLOCK 9000: CLOCK 5
```

Questi comandi riservano in memori delle spazioni dove possono essere immagazzinati i dati ricavuti al port 127, ed inizializza un puntatore,nella locazione di memoria 27000, a zero. Quindi viene indicata la linea a cui deve saltare il computer ogni volta che si raggiunga l'ora dell'allarme (linea 9000) e quindi viene attivato il sistema (CLOCK 5).

Regolate quindi l'allarme perchè "suoni" poco dopo la sua attivazione con:

CLOCK "Axxxxxx"

dove per xxxxxx subito dopo l'ora in corso. Ora date il RUN a un qualsiasi programma e la subroutine di allarme verrà eseguita ogni minuto raccogliendo ciò che c'è alla porta 127 e immagazzinandolo in una locazione di memoria indicata da"pointer"(questi dati non vengono utilizzati in questa routine). Se non avete nulla attaccato alle PORTS potete usare le linee 9040-9060 per fare qualsiasi altra cosa a intervalli regolari. Queste linee regolano l'allarme per un minuto dopo l'ora corrente prima di ritornare al programma principale.

Nota: alcuni accessori possono caricare le linee dei dati in maniera che una volta attivato CLOCK il sistema vada in crash (il sistema non il computer), in queste caso bisogna usare particolari accorgimen hardware oppure evitare di usare CLOCK.

DEF PROC nome della procedura
Tasto; 1 lo stesso di DEF FUN
Guardate pure: PROC,END PROC

Identifica una PROCedura che può essere richiamata con un nome(v.PROC). DEF PROC deve essere la prima parola chiave nella linea (può essere preceduta da spazi o da comandi per il controllo del colore). Il nome usato per la procedura deve seguire le regole per i nomi delle varibili: il primo carattere deve essere una lettera seguito da una stringa di lettere o numeri o "()". Gli spazi non sono significanti e lettere maiuscole o minuscule sono usate indifferentemente. Una procedura può avere lo stessa nome di una varibile senza creare alcun problema.

DELETE %numero di linea% TO %numero di linea%
Tasto : 7 lo stesso di ERASE

Rimuove dal programma tutte le linee dal blocco da voi indicate. Se viene omessa la prima cifra essa viene assunta come uguale al primo numero di linea dopo la linea 0, se omettete il secondo numero esso viene assun come uguale al numero dell'ultima linea del programma in meoria.

DELETE		TO	100	cancella tutte le linee dopo la linea 0 fino alla linea 100 compresa.
DELETE	100	TO		cancella tutte le linee dalla linea 100 compresa fino all'ultima linea del programma
DELETE	100	TO	100	cancella solo la linea 100
DELTE	0	TO	0	CANcella solo la linea 0
DELETE		TO		cancella tutto il programma esclusa la linea 100

L'ultimo esempio e molto diverso da NEW giacchè non effettua nessun CLEAR cosichhè le variabili rimangono tutte in memoria inalterate. Le linee indicate con DELETE deve essere presenti altrimenti otterrete il messaggio d'errore"U, No such line". DELETE PUò essere incluso in un programma con alcune limitazioni. Quando parti di un programma più in alto (numeri di linee più piccoli) vengono cancellati da un DELETE che fa

parte di una subroutine PROC o DO-LOOP il programma di solito si fermerà perchè gli indirizzi immagazzinati nello STACK non saranno più validi. Se un DELETE deve cancellare se stesso deve essere l'ultimo statement del blocco da cancellare. Una possibile funzione dei DELETE potrebbe essere quella di cancellare i DATA da un programma una volta che siano stati letti, così da liberare memoria(i numeri negli statements DATA occupano fino a 8 bytes per uno).

DO
DO WHILE condizione
DO UNTIL condizione
Tasto : D (WHILE è il tasto J, UNTIL è il tasto K)

DO e LOOP insieme con i loro qualificatori WHILE e UNTIL forniscono una struttura di controllo che ha alcuni vantaggi rispetto a quelli normalmente presenti nel BASIC. DO serve solo come un marker di riferimento a cui un concomitante LOOP fa tornare l'esecuzione di un programma:

```
10 DO
20 PRINT "Ciao": PAUSE 50
30 LOOP
```

questo programmino continuerà a scrivere sullo schermo "Ciao" finchè non verrà premuto il tasto BREAK.
DO può essere qualificato usando WHILE(qualche condizione).Se questa condizione è esatta tutti gli statements e le linee dopo DO verranno eseguiti mentre (WHILE) la condizione è vera. Se la condizione diventa falsa tutto ciò che segue DO verrà ignorato fino a che non verrà incontrato LOOP dallo STATEMENT subito dopo dicominciarà l'esecuzione del programma. DO UNTIL (qualche condizione) è esattamente il contrario. La parte di programma tra DO e LOOP verrà eseguita solo se la condizione è falsa(o altrimenti, è eseguito fino a che (UNTIL) la condizone non sia vera)

```
10 LET total=0
20 DO UNTIL total> 100
30 INPUT "introduci un numero";x
40 LET total = total + x: PRINT total
50 LOOP
60 PRINT "oltre 100"
```

La linea 20 potrebbe essere rimpiazzata con

```
20 DO WHILE total<= 100
```

DO LOOP multipli possono essere inseriti nello stesso modo di cicli FOR NEXT (Le coppie dell'esempio seguente sono fornite solo a spiegazione dei concetti sopraesposti ma non possono essere usati in un programma)

```
DO (A): DO (B): LOOP (B): LOOP (A)
```

questo esempio rappresenta una nidificazione semplice

```
DO (A): DO(B): LOOP(B) : DO (C): LOOP (C): LOOP (A)
```

quest'ultimo esempio indica due loop non nidificati dentro un loop più esterno.Poichè l'indirizzo DO è immagazzinato nello stack, non potete uscire da un loop DO-LOOP a meno che non usiate EXIT IF oppure POP altrimenti quest'indirizzo rimarrà sullo stack disturoando le altre

funzioni del computer. Se DO non è seguito da LOOP oppure se un DO condizionale cerca di sorpassare un loop e non ne riesce più a trovare la fine otterrete il messaggio di errore "S Missing LOOP".
Struttre di controllo molto simili a DO-LOOP esistono in altri computer sotto nomi differenti:

REPEAT (statements) UNTIL (condizione)

è equivalente a DO (statements) LOOP UNTIL (condizione)
e a

WHILE (condizione) (statements) END WHILE (o WEND)

è equivalente a

DO WHILE (condizione) (STatements) LOOP

DPOKE indirizzo,numero
Tasto: P
Guardate anche : FN P(indirizzo)

DPOKE singifica doppio POKE e l'equivalente in BASIC è:

POKE indirizzo, numero → INT(numero/256)x256
POKE indirizzo + 1, INT (numero/256)

In altre parole il byte meno significativo è a quell'indirizzo + 1. La maggior parte dei programmi in linguaggio macchina immagazzinano i numeri compresi tra 0 e 65535 in questo modo, inoltre molteX varianti di sistema hanno questo formato ed è quindi più facile alterarle con DPOKE senza ricorrere a calcoli. FNP (indirizzo) corrisponde al doppio PEEK.

EDIT %numero di linea%
Tasto: 0 (zero) (non è necessario entrare in modo grafico)

Non è la stessa cosa che premere SHIFT e 1. EDIT nel BetaBasic è una parola chiave e serva per aggirare la lunga sequenza: LIST (numero di linea), BREAK , SHIFT - 1. Perchè si possa considerare un miglioramento EDIT deve essere un tasto da premere da solo (senza alcuno shift), ma tutti i tasti delle lettere sono usati in questo modo e i tasti dei numeri servono per i numeri di linea. Comunque un numero di linea di soli non inizia con 0(zero) così può essere usato solo questo tasto. Il comando EDIT si ottiene solo se il tasto 0 è il primo tasto ad essere premuto dopo un ENTER. Se lo fate seguire da un numero di linea e premete ENTER la linea da voi indicata comparirà in fondo alle scherme. Se omettete il numero di linea comparirà la linea a cui è giàntoil cursore in quel momento. La semplicità dell'editing è stata migliorata permettendo ai tasti cursore su e giù di muovere il cursore entro la linea in fondo alle schermo da queste direzioni mentre prima non era possibile. Il cursore non si muoverà dentro le parole chiave ma si fermerà nello spazio seguente. Se proverete a muovere il cursore sopra o sotto la linea il cursore salterà alla fine della linea.

Nota: EDIT può essere ottenuto in graphic mode premendo il tasto 3

ELSE (statements)
Tasto: R

Fa parte della struttura IF-THEN. Di solito quando la condizione che segue IF è falsa, l'esecuzione del programma salterà alla linea seguente. Quando invece IF-THEN hanno associato più avanti nella linea il programma continua fino allo statement che segue ELSE. Quando invece la condizione dopo IF è vera la linea verrà eseguita solo fino a ELSE e quindi salterà alla prossima linea.
Per esempio:

```
10 PRINT "Dammi un numero":INPUT x
20 IF x=1 THEN PRINT "Vero": ELSE PRINT "Falso"
30 GO TO 10
```

Notate che ELSE è sempre preceduto dai due punti.
La situazione è più complicata se avete diversi IF THEN ELSE sulla stessa linea. Qui sotto vi sono alcuni esempi per mostrarvi quale ELSE sarà usato dai vari IF se le condizioni sono false:

```
IF-THEN (A): IF-THEN(B) : ELSE (B)
IF-THEN (A): IF-THEN(B) : ELSE (B) : ELSE (A)
IF-THEN (A): IF-THEN ELSE (A): IF-THEN (B): ELSE (B)
```

END PROC
Tasto : E
Guardate pure: DEFPROC, PROC
Indica la fine di una procedura con nome evitando al computer di eseguire gli statements compresi tra DEF PROC e END PROC a meno che la procedura non sia stata chiamata da PROC. In questo caso END PROC ne termina l'esecuzione un RETURN agli statements dopo PROC (guardate PROC PER avere più dettagli) . L'uso di END PROC senza DEF PROC provocherà la comparsa del messaggio di errore:

"Missing DEF PROC"

EXIT IF condizione
Tasto: I
GUARDAte pure : DO, LOOP

Fa parte della struttura DO-LOOP, per comprenderne il funzionamento dovete quindi leggere prima il capitolo che li riguarda. EXIT IF è usato per uscire da un loop DO-LOOP da qualsiasi punto che non sia un normale DO e LOOP. Se la condizione è vera, l'esecuzione del programma salta allo statement che segue LOOP, altrimenti non succede niente.
Esempio:

```
100 DO: PRINT "Linea 100": PAUSE 20
110 EXIT IF INKEY$= "STOP"
120 PRINT "Linea 120": PAUSE 20 : LOOP
130 PRINT "Fuori dal loop"
```

Queste loop continuerà a girare finchè non verrà premuto il tasto STOP
Notate che "Linea 120" non viene scritto quando uscite dal loop. Se non viene trovato LOOP otterrete il messaggio di errore"S,Missing LOOP"

GET variabile numerica o varibile stringa
Tasto: 8

Come INKEY$?GET è un modo di leggere la tastiera senza l'uso di ENTER. La differenza è che GET aspetta che un tasto venga premuto prima di andare avanti. Usato con una variabile stringa, GET produce una stringa di un carattere:

```
10 GET A$: PRINT A$: GO TO 10
```

Questo programma funzione come una macchina da scrivere. Potete anche passare dalle lettere maiuscole a quelle minuscole oviceversa nel solito modo. SHIFT-5 farà tornare il cursore indietro di uno spaziosenza cancellare e ENTER è equivalente a PRINT. Una versione + sofisticata del programma qui sopra potrebbe essere:

```
10 GET A$: PRINT A$; FLASH 1;"B";FLASH 0;" ";CHR$ 8;CHR$ 8;: GO TO 10
```

Se GET è usato con una variabile numerica(GETx o GET tasto) la variabile sarà uguale a 1 se viene premuto il tasto 1, uguale a 2 con il tasto 2, e così via fino al tasto 9. Quindi il valore sarà 10 per il tasto "A", 11 per il tasto "B" e cosi via. GET è particolarmente utile nel menù dei programmi (guardate ON),

KEYWORDS 1 e0
Tasto: 8

Controlla se vengono visualizzati UDG e le parole chiave del BetaBasic. All'inizio il sistema è nello stato KEYWORDS 1, quando volete usare gli UDG eseguite KEYWORDS 0. Inoltre ciò vi permetterà di rimanere in graphic mode per quanto tempo volete mentre nello stato KEYWORDS 1 ogni volta che premete un tasto tornate nel modo normale. Sebbene il listato può sembrare confuso il programma funzionerà normalmente. (KEYWORDS è l'unico tasto chiave che non può essere disttivato, il tasto grafico corrispondente è SPACE)

LOOP
LOOP WHILE condizione
LOOP UNTIL condizione
Tasto: L (WHILE ha il tasto J,UNTIL ha il tasto K)
Guardate pure: DO,EXIT,IF

Fa parte della struttura DO-LOOP, fa si che l'esecuzione del programma torni indietro al corrispondente DO. I qualificatori WHILE e UNTIL permettono di effettuare il loop condizionate.
LOOP WHILE (condizione) effettua il loop solo se la condizione è vera altrimenti esegue lo statement sub$to dopo LOOP. LOOP UNTIL è il contrario, il loop non è effettuato se la condizione è verà ed è effettuato se è falsa. Se usate il LOOP senza il corrispondente DO otterrete il messaggio"T, LOOP without DO".

ON
Tasto: O
Come in:
GO TO e GOSUB ON numero; numerolinea, numero linea;

ON permette di eseguire il GOSUB e il GOTO a una linea seconde il numero che si trova sunito dopo ON. Una sintassi più usuale sarebbe ON numero GOTO num.linea,num.linea, ecc... ma ciò è reso difficile dalla struttura della tastiera dello SPECTRUM. On è più flessibile dell'usuale alternativa delle SPECTRUM:

```
10 INPUT numero: GO TO numero*100+100
```

perchè i numeri di linea non devono essere sistemati in un sequenza:

```
10 INPUT numero: GO TO numero;  90,135, 60, 40
20 PRINT"Introduci un numero da 1 a 4": GO TO 10
```

Nell'esempio sopra se la variabile "numero " ha un valore 1 viene eseguito un GO TO 90, se è 2 viene eseguito un GO TO 135 e cosi via.Se numero viene fornito negativo viene prima positivizzato quindi utilizzato.
Se "numero" non è compreso tra 1 e 4 il programma continua con là statement successivo, che in questo caso specifico forza l'utente a ritentare.
INPUT potrebbe essere sostituito da GET PER UNa implementazione molto elegante di un menu.

ON ERROR numero di linea
Tasto: N

Dopo l'esecuzione di qusto comando ogni volta che si verifica un errore viene eseguito un GOSUB automatico alla linea specificata. Ciascuno dei messaggi indicati nel manuale dello SPECTRUM e del BetaBasic tranne "0 Ok" e "9 Stop in INPUT" provocheranno un tale tipo di GOSUB.
Il comando è inattivato da ON ERROR0, ma è inattivato anche durante l'uso della subroutine d'errore e viene riattivato dopo il RETURN al programma principale(una routine d'errore che si chiama da se crerebbe della confusione). La subroutine d'errore può usare tre variabili particolari che non sono parole chiave e devono quindi essere battute per intero e non possiedono un unico tasto che le richiama.. LINE e STAT indicano la linea e lo statement dove è avvenuto l'errore e ERROR indica il codice di errore, ivalori da 1 a 8 sono considerati come tali mentre le errore A darà ERROR = 10, ERROR=11 sarà corrispondente all'errore B.
E' preferibile che la vostra subroutine consideri non più di uno o al massimo due messaggi d'errore. Qui di seguito vi viene offerto un esempio, che potrebbe essere solo una parte di un programma, in cui vengono plottati dei punti sullpo schermo e ogni volta che uno di questi è fuori schermo viene semplicemente eliminato:

```
100 ON ERROR 5000
110 FOR n=1 TO 10: INPUT "x coord";x;"y coord";y
120 PLOT x,y: NEXT n

4990 STOP
5000 IF error = 11 AND line =120 THEN RETURN:ELSE POP:CONTINUE
```

Nota: lo STOP è stato usato per evitare che la subroutine venga esegiita

per errore. Vengono controllati entrambe LINE ed ERROR perchè "B Integer out of range" è un messaggio piuttosto comune. Il TERUAN al programma principale avverrà allo statement successivo a quello che ha causato l'errore, così ritornerà a NEXT n. Se non si verifica quell'errore alla linea indicata vieneusato il CONTINUE. Il risultato sarà che CONTINUE fa eseguire lo statement (a meno che non sia stato un"BREAK into program") e siccome CONTINUE non riattiva ON ERROR viene visualizzato il messaggio d'errore. POP elimina l'indirizzo di ritorno al programma principale che altrimenti rimarrebbe nello stack del computer a meno di un CLEAR 0 RUN. Un errore da maneggiare in maniera un po' differente è il "BREAK into program". Siccome ON ERROR viene disattivato durante l'esecuzione della subroutine, accade che il programma della subroutine d'errore venga fermato perchè voi state ancora premendo il tasto di BREAK. Ad ogni modo se volete che venga eseguita una subroutine durante un BREAK dovete introdurre un ritardo al primo statement della subroutine di errore così che abbiate il tempo di interrompere la pressione sui tasti. BEEP va molto bene, lo potete usare con una frequenza quasi inaudibile. Per esempio:

```
  100 ON ERROR 5000
  110 PRINT"prova";: PAUSE 10: GO TO 110

 4990 STOP
 5000 IF error =21 THEN BEEP 1,69: BORDER RNDx7:RETURN:ELSE POP:
      CONTINUE
```

Se dimenticate di limitare l'uso della subroutine d'errore a particolari messaggi d'errore potreste anche entrare in loop senza fine. Un esempio potrebbe essere un programma in cui voi vi aspettate di uscire usando "STOP in INPUT". In questa situazione il sistema particolare di BREAK del BEta Basic funzionerà se l'avete attivato altrimenti dovrete spegnere il computer.

PLOT coordinata x, coordinata y %;stringa%
Tasto: il normale tasto per PLOT non in modo grafico

Come potete vedere dalla sintassi sopra indicata, Beta Basic permette di plottare una stringa come una qualsiasi serie di pixel. Le coordinate di PLOT si riferiscono allo schermo partendo dall'angolo in alto a sinistra dello schermo. La stringa può essere lunga fino a 32 caratteri, se è più lunga non viene stampata. Possono essere usati tutti i normali qualificatori di PLOT (INVERSE,INK,OVER....). La stringa non deve contenere il CHR$ 13 oppure una parte della stringa non verrà ripertata. Se la stringa esce dal lato destro dello scerno il rimanenet ricomparirà sul lato sinistro. Se invece dovesse superare il limite inferiore e superiore dello schermo comparirà il messaggio "Integer oo$ of range". La posizione di plot che viene usata da DRAW come punto di partenz per tracciare una line non viene alterata dal plottaggio di una stringa. Cambiando le coordinate di plottaggio di un carattere si possono ottenere degli effetti di spostamento molto più fine che con PRINT AT:

```
100 FOR x=16 TO 224: PLOT x,x/21;"()":NEXT x
```

Per ottenere una velocità maggiore provate ad aggiungere STEP 2 o 3. Siccome i segni "()" hanno un bordo libero di almeno un pixel senza niente durante il movimento (solo di un pixel per volta) provoca l'automatica cancellazione della posizione precedente. Alcune lettere come "T" hanno i pixels di INK che si estendono sino ai lati del quadrato di8X8 che occupa ciascun carattere, questo significa che spostandole in alcune direzioni reimarrebbero una serie di tratti, bisogna quindi che voi in questi casi provvediate a cancellare la vecchia posizione prima di stampare la nuova. Se state disegnando dei caratteri vostri è una buona norma cercare di disegnarli lasciandogli una cornice interna di almeno un PIxel dove vi sia solo PAPER. E' molto utile poter plottare una stringa con didascalie di grafici e diagrammi. Il seguente esempio mostra come questo comando renda capace lo SPECTRUM DI produrre sopra e sotto scritte/

```
100 LET A$=" Lo sai quante molecole ha/cm+3-Na-2+SO-4+"
110 LET X = 0: LET Y = 140
120 FOR C= 1 TO LEN A$
130 IF A$(C) = "-" THEN LET  Y=Y-3: NEXT C
140 IF A$(C) = "+" THEN LET Y=Y+3:NEXT C
150 PLOT X,Y;A$(C): LET X=X+8
160 IF X >= 248 THEN LET X=0: LET Y=Y-12
170 NEXT C
```

Il programma usa i simboli + e - come codici di controllo per spostare la posizione di PLOT più in altro o più in basso di 3 pixels. In un vostro programma sarebbe comunque meglio che usiate altri simboli perchè questi possono provocare spostamento di PLOT non desiderato. Provate ora a modificare nella linea 150 la variazione di X in modo che sia diversa da 8, in questo modo variarete la spaziatura tra le lettere. Con spaziature molto strette potreste usare PLOT OVER 1 in modo che la PAPER della lettera scritta dopo xxx si sovrapponga e non cancella la lettera scritta prima.Unaltro uso potrebbe essere l'introduzione di spaziature nel testo in un Word Processing così che ogni riga contenga un numero intero di parole.

POP %variabile numerica%
Testo= 9

Rimuove un indirizzo dallo stack di GOSUB/ DO-LOOP/RROC. Il numero di linea indicato è assegnato a una variabile numerica. Questocomando permette di uscèee da un loop DO-LOOP senza lasciare indirizzi inutili nello stack. POP usato da solo elimina semplicemente gli indirizzi, ma POPloc rende reperibile il valore del numero tolto dallo stack sotto il nome della variabile LOc. Così la subroutine o la procedura sanno da dove sono state chiamate. Sfruttando questo meccanismo si può effettuare una specie di RETURN anche se non è proprio la stessa cosa visto che non è possibile psecificare con questo meccanismo lo statement:

```
100 GOSUB 500
110 STOP
500 POP loc
510 PRINT "Subroutine chiamata dalla linea";loc
520 GO TO loc+1
```

Se rimpiazzate la linea 510 con RETURN otterrete il messaggio "RETURN without GOSUB" poichè non c'è più l'indirizzo ri ritorno sulle stack. L'uso di POP quando non vi sono dati sulèe stack fa comparire il messaggio:" No POP data"

PROC nome
Taste: 2 (lo stesso di FN)
Vengono qui discussi: DEF PROC, END PROC

PROC è l'abbreviazione di procedura. Sono molto simili all GOSUB ma hanno il vantaggio che voi non dovete andare a cercare, durante la programmazione, in quale parte del programma si trova la subroutine vhe vi serve, il computer la troverà ovunque sia nel listato, facendo si che DEF PROC sia il primo statemnt nella linea (questa limitazione accelera lo svolgimento del programma). Il nome della procedura può essere un qualsiasi nome di varibile anche già presente in memoria, non importa se vi sono degli spazi e le lettere sono maiuscole o minuscole.
La procedura può essere lunga un numero qualsiasi di linee e viene chiusa da END PROC che può essere sistemato in qualsiasi posizione.
Come DEF FN, DEF PROC viene ignorato se non chiamato da PROC, e l'esecuzione del programma passa oltre. La procedura ha a disposizione tutte le variabili del programma principale e qualsiasi variabile creata e alterata dalla procedura è a disposizione del programma principale.

```
10 FOR n=1 TO 20: PROC dis.quadr.: NEXT n
200 DEF PROC dis.quadr.: LET lato=RND x 20 + 20
210 PLOT RNDx215,RNDx135: DRAW lato,0: DRAW 0,lato
220 DRAW -lato,0: DRAW 0,-lato: END PROC
230 PRINT " Finito "
```

Un programma strutturato "ideale" potrebbe consistere in una serie di procedure, ciascuna delle quali fa un lavoro particolare e può essere controllata separatamente. Queste sono quindi chiamte dal programma principale che potrebbe essere una cosa tipo:

```
100 PROC prep. scherme
110 PROC gioca
120 PROC vis. punti: STOP
```

Se non avete definito la procedura l'uso di PROC provocherà la comparsa del messaggio d'errore "Missing DEF PROC". Accadrà la stessa cosa anche se usate END PROC senza DEF PROC(l'eccezione si verifica qualora sullo stack sia presente l'indirizzo di un RETURN,- END PROC non può sapere che non è un indirizzo derivato da un PROC e quindi l'esecuzione salterà alla linea indicata). Se vi dimenticate di includere END PROC il programma principale si fermerà con il messaggio di errore "X No END PROC"

quando cercherà di saltare al di la della procedura non chiamata e non ne troverà la fine.

RENUM %inizio TO fine% %LINEnuova partenza% %STEP intervallo%
Tasto: 4

RENUM usato da solo rinumera tutto il programma cosi che la linea 10 sia la prima linea del programma e le altre si susseguono con un intergallo di 10. Può essere specificato un blocco di lineee su cui effettuare la rinumerazione nel modo seguente:

```
RENUM (130 TO 220) rinumera le linee comprese tra 130 e 220
RENUM (130 TO)     rinumera le linee da 130 in poi
RENUM ( TO 100)    rinumera tutte le linee,esclusa la linea Ø?
                   dall'inizio fino alla 100
```

Se impossibile rinumerare le linee del blocco da voi indicato senza che i nuovi numeri si estendano nei numeri di linea al di fuori del blocco comparirà il messaggio d'errore "B No room for line", che in questo contesto non ha lo stesso significato che gli viene attribuito dal manuale Sinclair. Può essere scelto un nuovo numero di linea al di fuori di 10 usando contemporaneamente LINE(la normale parola chiave). Se lo desiderate potete anche alterare l'intervallo che la rinumerazione introdurrà tra le linee con STEP. Questi ultimi due qualificatori possono essere omessi ma se usati devono essere introdotti seguendo un certo ordine:

```
RENUM
RENUM LINE 100 STEP 20
RENUM ( 1540 TO) LINE 2000
RENUM (100 TO 176) LINE 250 STEP 5
```

Quando viene fatto partire l'ordine viene prima di tutto controllato il programma per vedere se esistonoespressioni tipo "GO TO 1 x 100" o simili. Tali espressioni potrebbero riferirsi a linee che debbono essere rinumerate e sono troppo difficili per qualsiasi normale subroutine di rinumerazione, così RENUM si interrompe con lascritta "Too hard" e vi da l'indicazione della linea e dello statement a cui si trova l'espressione. Vi da quindi la possibliità di cambiare l'espressione o con ON o mettendola temporaneamente tra virgolette PRINT "GO TO 1 x 100 ". Una volta che modificato tutte le espressioni fate ripartire RENUM. Vengono modificati riferimenti al blocco di linee modificato in tutto il programma, compresi GO TO, GOSUB,RESTORE, RUN, ON, ON ERROR, TRACE, LIST?, LLIST, LINE, DELETE. Non vengono modificati i CLOCK che quindi dovrete fare da soli ammano.

Nota: RENUM crea una tabella dati nella memoria di schermo che viene cancellata appena il RENUM è completato.

ROLL direzione ;% x coord, y coord; largh, lungh%
Tasto : R
Guardate pure : SCROLL

ROLL muove tutto lo schermo o una porzione di esso in una della 4 direzioni di un pixel. Qualsiasi cosa ~~xxxxxx~~ scompaia da un lato della finestra considerata ricomparirà dal lato opposto. In altre parole il comando non distrugge niente di ciò che è presente sullo schermo lo altera semplicemente (diversamente da SCROLL). La direzione di ROLL è indicata usando i numeri 5, 6 ,7 ,8 subito dopo il comando corrispondendo ciascuno di questi come spostamento alla freccia indicata sopra essi. Siccome l'uso di ROLL per solo una volta provoca solo un piccolo spostamento è meglio che il comando venga usato in un loop. Disegnate qualcosa sullo schermo e fate un list e quindi provate:

```
100 FOR d= 5 TO 8: FOR p = 1 TO 100
110 ROLL d: NEXT p: NEXT d: STOP
```

Uno spostamento in diagonale può essere ottenuto spostando prima di un pixel a sinistra e quindi in su (a seconda della direzione voluta si possono accostare vari spostamenti). Una specifica porzione di schermo può essere spostata facendo seguire all'indicazione della direzione 4 parametri: x e y sono le coordinate dell'angolo in alto a sinistra della finestra considerata (usate lo stesso sistema di coordinate che usate per PLOT E DRAW) quindi la larghezza della finestra (questa volta espressa in numero dicaratteri) quindi la lunghezza della finestra espressa in pixels. La largheezza può essere tra 1 e 32 e l'altezza da 1 a 174. ROLL potrebbe essere molto utile per creare effetti di movimento fine nei giochi. La routine qui sotto creano alcuni effetti interessanti:

```
100 LIST : LIST : LIST
110 ROLL 5 ; 0 , 175;32,88: ROLL 6; 0,175;16,174
120 ROLL 8;0,87;32,88: ROLL 7; 128,175; 16,176
130 GO TO 110
```

oppure

```
200 FORN=1 TO 7 : LIST: NEXT N
210 FOR L=1 TO 175 : ROLL 7;128,175;16,176 / NEXT L
```

SCROLL %direzione%;%x coord.,y coord.; profondità , lunghezza%
Tasto: S
Vedete pure : ROLL

SCROLL ha un sintassi molto simile a ROLL. Una differenza è che SCROLL può essere usato da solo (senza qualificatori) nel qual case muove tutto lo schermo verso l'alto di un carattere (come lo ZX 81). Se SCROLL è seguito da 5,6,7,8 tutto lo schermo viene mosso di un pixel nella direzione indicata da questi numeri (vedete ROLL). Qualsiasi cosa venga spinta fuori dallo schermo viene perduta. Lo schermo che compare dall' altro lato è pulito. E' possibile selezionare una certa area di schermo da far scorrere se all'indicazione di direzione si fanno seguire altri dati:"x"e"y" che sono le coordinate dell'angolo in altro a sinistra della pomzione interessata, la larghezza in caratteri e la lunghezza inpixel.

Sia ROLL che SCROLL sono molto utili nel creare giochi o effetti grafi ci di qualsiasi tipo. Provate ad applicare SCROLL negli stessi esempi forniti dan ROLL e confrontate i risultati. Eccovi un programma che ac cetta una stringa in INPUT e la fa scorrere attraverso lo schermo:

```
100 [illegible] INPUT A$
110 FOR C=1 TO LEN A$
120 PRINT AT 10,31;INK 7; A$(C)
130 FOR P=1 TO 8: SCROLL 5;0,95;32,8: NEXT P
140 NEXT C
150 FOR P = 1 TO 255: SCROLL 5;0,95;32,8:NEXT P
```

a stringa viene stampata una lettera per volta nello stesso posto con l'INK dello stesso colore della PAPER. Questo fa si che ciascunalette ra appaia lentamente come se venisse fatta scorrere via dalla posizione in cui era stata scritta con inchiostro invisibile. Il loop interno FOR NEXT muove il carattere a sinistra di una posizione,prima che il seguen te venga scritto,usando una finestra di un carattere solamente. La li- nea 150 muove la stringa fuori dallo schermo una vnàta che è stata stampa ta; oppure potreste aggiungere alla stringa 32 spazi, oppure potreste trovare utile usare il PLOT del Beta Basic sostituendo la linea 120 con:

```
120 PLOT 248,95;A$(C)
```

lo stesso principio potrebbe essere usato per presentare un teste qual siasi (es: le istruzioni del programma):

```
200 DATA "TANTO TEMPO FA, IN UNADISTANTE..."
210 DATA "BLA BLA BLA BLA BLA"
300 FORL=1 TO 2: READ A$
310PRINT AT 21,0;INK 7; A$
320 FOR P = 1 TO 8: SCROLL 7: NEXT P: NEXT L
330 FOR P = 1 TO 176: SCROLL 7: NEXT P
```

SORT array stringa o array numerico o stringa
Taste: M

SORT sistema stringa , numeri o lettere in ordine ascendente o discen- dente. Verrà prima di tutto discusso il suo us e con gli string array. Ecco un programma per generare un array di 100 stringhe di 10 lettere:

```
100 DIM A$(100,10)
110 FOR S= 1 TO 100: FOR L =1 TO 10
120 LET A$(S,L)= CHR$(RND x 25+65)
130 NEXT L: NEXT S: GO TO 200
140 SORT A$
200 FOR S = 1 TO 100: PRINT A$(S): NEXT S
```

Appena creato l'array, il che richiede un po' di tempo, verrà stampato. Ora date GO TO 140 e l'array verrà stampato di nuovo ma questa volta se conde un certo ordine (non date RUN 140 o perderete l'array). Il tempo che SORT ci metterà a mettere ordine nell'array è circa 1/5 di secondo

questo tempo aumenterà in maniera meno evidente se usate stringhe lunghe, l'incremento di tempo sarà più sostanzioso se usateinvece un numero maggiore di stringhe(per 200 stringhe impiegherà circa 0,7 sec., per 400 circa 3 sec.).
Le stringhe sono ordinate secondo il loro CODE, e per questi riferitevi alla lista presente sul manuale del Sinclair. Se cambiate la linea 140 con SORT INVERSE A$ l'ordine sarà inverso. Possiamo anche selezionare un blocco di stringhe da ordinare con:

SORT A$(1 TO 20)

che ordinerà solo le prime 20 stringhe e

SORT A$(30 TO)

che ordinerà dalla stringa 30 in poi. E' pure possibile ordinare una parte particolare della stringa:

SORT A$()(2 TO)

che ordinerà l'intero array sulla base della seconda lettera di ciascuna stringa, la prima lettera non verrà tenuta in considerazione ma verrà spostata insieme al resto della stringa. Con SORT è possibile sviluppare un data base veloce e flessibile. In questo contesto èpiù comune chiamare l'array"file" e la sua stringa "record". Le aree di ciascuna stringa verrebbero probabilmente riservate per particolari tipi di informazione e verrebbero chiamate " campi". Una stringa molto lunga potrebbe servire immagazzinare dati di diverso tipo, ad esempio si potrebbe far si che una stringa del genere immagazzini nei prime 20 caratteri il nome di una persona, nei successivi 20 il suo indirizzo e l'ultimo carattere la sua età. Questo perchè l'età di una persona è molto probabile che abbia valori compresi tra 0 e 255! Si potrebbe quindi far ricorso a una linea del genere:

LET A$(9,41)= CHR$ età

Un tale sistema per conservare numeri è semplice e risparmia un sacco di memoria. Adesso supponete di dover memorizzare qualcosa di più complesso come un bilancio bancario. Se usate:

LET A$(9,41 TO 48)= STR$ bilancio

l'informazione verrà sistemata nella stringa ma sarà allineata a sinistra(così il "9" di un bilancio di 9 sterline si verrà a trovare allineato con "1" di un bilancio di 100 sterline). Questo impedisce di lavorare correttamente in questo campo. La risposta è facile, basta formattare la cifra in modo di allinearla a destra e ciò è possibile con PRINT US:

LET A$(9,41 TO 46)= (PRINT US "000,00", bilancio)

Per comprendere meglio il funzionamento di questo statement dovete guardare sotto PRINT U$ e USING, inoltre guardate anche PRINT C$ che vi aiuterà a risparmiare memoria nell'immagazzinare numeri. Potete ora ordinare una serie di records secondo l'età; il bilancio bancario e l'ordina alfabetico. Notate che siccome "1" nella tabella dei CODE$ viene prima di "2" SORT sistemerà i numeri piccoli prima di quelli grossi e SORT INVERSE farà esattamente l'opposto. Questo avviene solo quando i numeri sono rappre

sentati sotto forma di stringa. E' necessario essere molto precisi con questi array (bisogna far si che la prima lettera di un certo dato occupi sempre un certo poste nella stringa).
SORT può anche lavorare su stringhe semplici e array modo dimensionale:

```
INPUT S$: SORT S$: PRINT S$
```

Se introducete Fred Bloggs vi darà come risultato "BFdegglors". Usato in questo modo non sembbra molto utile ma permette a SORT di lavorare su un certo tipo di dati numerici chepuò essere memorizzato più efficentemente in stringhe , come per esempio:

```
LET S$(posizione)= CHR$(dato)
```

SORT lavora pure con arrays numerici convenzionali di una o due dimensioni, usando la stessa sintassi che si usa per gli array stringa. Un array numerico a due dimensioni può essere considerato come una tavola sulla quale la prima dimensione indica la fila la seconda la colonna:

```
SORT B( 1TO 20) (2)
```

rdina le prime 20 linee dell'array numerico B sulla base del numero presente nella seconda colonna di ciascuna fila(le file di numeri verranno comunque rimosse in toto all'atto del loro spostamento). Notate che è necessario usare sempre uno slicer (qualcosa tra parentesi) per differenziare l'array B() dalla variabile semplice B. Le operazioni di SORT agli array numerici sono circa 4 volte più lente di quelle sulle stringhe perchè SORT deve controllare due diversi formati di numero (pag.170 EL manuale Sinclair) e la possibilità di numeri positivi e negativi. Siccome non è un ordinazione sulla base dei CODE$ i numeri più grandi verranno prima di quelli più piccoli. Anche qui il discorso può essere invertito con SORT INVERSE.

TRACE numero di linea
Esempio: T

molto utile de debugging dei programmi BASIC, può indicare la linea esecuzione, lo statement, variabili da voi indicate esplicitamente riduce la velocita di esecuzione del programma di quando volete sino dipendere dalla pressione di un tasto l'esecuzione di un singolo atement per volta. TRACE PROVOCA LA chiamata di una subroutine prima vnega eseguito ciascuno statement. La subroutine ah a disposizione variabili speciali (non sono parole chiave) LINE e STAT che indicano numero di linea e di statement della parte di programma che deve eseguito. TRACE viene disattivato durante l'esecuzione della subtine da esso chia mata e viene riattivato dal RETURN alla fine della ssa. Il contenuto minimo della subroutine deve essere:

```
9000 PRINT INVERSE 1;line;":";stat:  RETURN
```

Aggiungete questo statement (TRACE 9000) nel programma nel punto da cui vobete iniziare il debugging del programma stesso. Se volete disattivare TRACE aggiungete dove volete che finisca la sua azione TRACE Ø. L'uso della routine fornitavi qui sopra vi darà il listato delle linee e degli statements a seconda dell'ordine di esecuzione mentre questi vengono eseguiti, INVERSE serve invece per distinguerlo da qualsiasi altra cosa presente sullo schermo. RUN, CLEAR e TRACE Ø disattivano TRACE. Se volete rallentare l'esecuzione del programma che state correggendo potete aggiungere uno statement PAUSE nella subroutine di TRACE, potete anche usare PAUSE Ø per far eseguire un solo statement del programma ogni volta che premete un taste. Possono essere anche visualizzate delle variabili e quindi il loro variare con lo scorrere del programma, ma per far queste è bene che queste variabili siano dichiarate subito all'inizio del programma altrimenti si potrebbe ottenere un "Variable not found". Per evitare confusione tra l'autput su schermo di TRACE e l'esecuzione del programma che potrebbe ~~dex~~ creare un suo output simile, si può usare un PRINT AT, ma sarà necessario ripristinare le vecchie coordinate dopo l'uso altrimenti si potrebbe alterare l'esecuzione del programma:

```
9000 LET col=PEEK 23688: LET fila= PEEK 23689
9010 PRINT AT 0,0;line;":";stat;"     .","A$ =";A$;"       "
9020 POKE 23688,col:POKE 23689;fila: RETURN
```

Questo programma stampa la linea, lo statement e la variabile A$ nella prima linea in alto dello schermo (gli spazi aggiuntipermettono di cancellare altri caratteri gia presenti sulla stessa linea in maniera di evitare sovrapposizione). Oltre a ciò memorizza le coordinate di PRINT prima dell'esecuzione della subroutine di TRACE e le risistema come prima cosi da non interferire con l'autput su video del programma principale.

UNTIL condizione
Fatto: NK
Come in: PRINT USING formato stringa, numero
Viene qui trattato anche l'uso di : FN U$

Sia USING che FN U$ permettono la formattazione di un numero che deve essere stampato. In tutte le sue possibili applicazioni USING può essere rimpiazzato da FN U$. FN U$ vi ritorna la stringa che PRINT USING stamperebbe, questo permette la formattazione dei numeri per l'uso con LET, PRINT e qualsiasi altra comando che possa essere applicato con una stringa. Con USING o FN U$, il formato desiderato viene specificato da una stringa nella quale l'asterisco sta per gli spazi e gli zero stanno per lo zero, ed entrambe possono essere usati per indicare il numerà di cifre dopo la virgola:
(per %&vedi asterisco)

```
100 FOR N=1 TO 20 : LET X = RND x 100
110 PRINT X,USING "&&&.&&": NEXT N
```

Notate come sia più chara la rappresentazione dei numeri dopo la formattazione. Confrontando le due colonne potete vedere che USING arrotonda alla cifra più vicina. Provate ora diversi formati con il numero 12,3456

"&&,&"	12,3
"&&&,&"	&12,3
"&&&,&&"	&&12,35
"000,00"	012,35
"00"	12
"£00,00"	£12,35
"0,00"	%__3

Il penultimo esempiomostra come sia possibile includere altri caratteri oltre l'asterisco e lo zero. L'ultimo esempio mostra un uscita con un "%" che indica un insufficienza del formato a contenere il numero.
Nota: USING non funziona con notazioni scientifiche.

FN U$ viene usato nello stesso modo ma invece di:

PRINT USING A$;numero

dovete dare:

PRINT FN U$ (A$,numero)

WHILE condizione
Tasto: 7

Permette l'esecuzione condizionata di DO e LOOP, riferitivi a questi comandi per ulteriori dettagli.

XOS, XRGx, YOS, YRG

Queste quattro parole non sono parole chiave, sono un tipo speciale di variabile che permettono di cambiare la sede e l'origine usate dai comandi PLOT, DRAW e CIRCLE.
XOS sta per la regolazione dell'asse delle x, YOS sta per l'asse delle y. XRG sta per range x axis, YRG sta per RANGe y axis. Un modo in cui sono speciali è che queste variabili sono regolate a ZERO e non cancellate quando viene dato un RUN o un CLEAR. Quindi se dope PRINT XOS dope uno di questi comandi otterete "Ø" e non "Variable not found". Gli offset hanno valore zero fino a che non vengone ridefiniti con un LET. RUN e CLEAR li riportano a zero. Per cambiare la localizzazione della origine è molto più semplice farlo con i LET che usare molti statement PLOT:

LET XOS= 128: LET YOS= 88

SPOgterà l'origine al centro dello schrmo e vi permette di plottare valori tra -127 e +128 e valori di y compresi tra -86 e +87.

XRG normalmente ha il valore 256 (potete plottare su 256 differenti posizioni lungo l'asse delle x) mentre YRG è normalmente 176. Cambiandone i valori cambia la scala su cui fare i vari PLOT e DRAW:

```
10 GO SUB 100: REM normale
20 LET XRG = 128: GOSUB 100
30 LET YRG = 88: GO SUB 100
40 LET XRG = 256 : GO SUB 100: STOP

100 CLS: PLOT 0,0: DRAW 50,0: DRAW 0,50
110 DRAW -50,0: DRAW 0,-50: PAUSE 100: RETURN
```

Il prime quadrato disegnate sara normale, successivamente verrà distorto lungo l'asse dell X poi su entrambe gli assi X e Y, poi solo su quelle dell' Y, ciò avviene perche DRAW segue la variazione di scala che si effettua alterando YRG e XRG. Il seguente esempio mostra un onda sinuseidale cambiando sia l'off-set che la scala:

```
100 LET XRG= 2xPI: REM 360°
110 LET YRG= 2.2: REM Il seno varia tra +1 e -1
120 LET YOS= 1.1: REM Origine dell'off-set a metà schermo
130 FOR N=0 TO 2xPI STEP 2xPI/256
140 PLOT N,SIN N: NEXT N
```

Notate che il valore di off-set segue i valori di scala. Il punto finale di arrivo di una curva viene piazzato seguendo la variazione di scala ma la curva disegnata non verrà distorta. Allo stesso modo , sebbene il centro di un cerchio verrà piazzato secondo le coordinate calcolate sulla nuova scala, il raggio non viene alterato da XRG e YRG e inoltre non è possibile creare cerchi distrorti agendo sulle suddette variabili.

F U N Z I O N I

GENERALE

Sono state aggiunte dieci nuove funzioni al BetaBasic usando la definizione di funzione alla linea Ø (che di solito non è visibile) per puntare alla memoria alta dove risiede il L/M che fa il vero lavoro. Queste funzioni non funzioneranno se viene effettuato un tentativo di usarle senza la parte in L/M (provocherà il crash del sistema o come si dice il sistema si "impunta"). D'altra parte se non è presente la linea Ø comparirà la scritta "FN without DEF" quando tenterete di usarle. In quest'ultimo caso il BetaBasic funzionerà perfettamente ma averte perso solamente la capacità di usare le nuove funzioni. Se salvate su cassetta un programma scritto metre il BetaBasic è residente (anche la linea Ø) una copia della linea Ø verrà salvata con il programma e in caso di reload sarà presente. Se caricate con LOAD un programma che non stato scritto con il BetaBasic mentre questo è residente cancellerà

la linea Ø. Per ovviare a questo inconveniente liberatevi del programma in memoria con NEW che nel BetaBasic non cancella la linea Ø e quindi fate il MERGE con il nuovo programma. Potete rimuovere la linea Ø se lo desiderate,magari per risparmiare memoria, facendo DELETE Ø TO Ø. Le funzioni sono elencate in ordine alfabetico.

FN C$ (numero)
Guardate pure : FN N(stringa)

Questa funzione converte un numero intero (compreso tra o0 e 65535) in una stringa a due caratteri permettendo la memorizzazione con un notevole risparmio di memoria. L'equivalente in Basic è:

```
LET A = INT(numero/256): LET B = numero - Ax256
LETC$ = CHR$ A + CHR$ B
```

Se provate a stampare la stringa risultante otterrete abbastanza spesso il messaggio "K Invalid colour", poichè può capitare che siano dei codici di controllo e quindi non stampabili. FN N(stringa a due caratteri) verrà usato per ritrasformare la stringa di due caratteri in un numero normale. Poichè vengono usati solo 2 bytes per numero contro i 5 del sistema normale, vale quinei la pena di considerare l'uso di questa funzione quando si hanno molti dat da immagazzinare. I dati non è necessario che siano numeri interi. Ad esempio per ottenere una precisione accettabile con un numero tipo 87,643, lo si può moltiplicare per 100 in mòdo da ottenere 8764,3 la cui parte intera può essere usata da FN C una volta che debba essere riutilizzata la si richiama con FN N e poi la si divide per 100 ottenendo 87,64 che per la maggior parte dei calcoli rappresenza una precisione abbastanza accurata. Il seguente esempio mostra l'implementazione con un array:

```
100 DIM A$(500,2)
110 FOR E = 1 TO 500: LET A$(E) = FN C$(Ex10): NEXT E
120 PRINT " Ecco l'array create ; premi un tasto per stamparlo"
130 PAUSE Ø
140 FOR E = 1 TO 500: PRINT E, FN N(A$(E)): NEXT E
```

Notate che questo array usa solo 1K contro i 2,5K che userebbe un array standard con lo stesso numero di elementi. Il comando SORT funzionerà perfettamente su questo ARRAY.

FN H (stringa)
Guarda pure: FN H$(numero)

Questa funzione fornisce l'equivalente decimale di una stringa da 1 a 4 caratteri che rappresenti un numero esadecimale valido. Non è importante se le lettere siano maiuscole o minuscole:

```
FN D ("FF") = 255
FN D ("10") = 16
FN D ("4000")= 16384
FN D ("e") = 14
```

Per introdurre dei numeri in memoria potreste usare (se avete i codici esadecimali);

```
INPUT A$: POKE indirizzo,FN D(A$)
```

L'uso di una stringa nulla con più di 4 caratteri e con caratteri che non rappresentino valori esadecimali provocherà il messaggio:

"Invalid argument"

FN H$(numero)
Guarda pure: FN B$(stringa)

L'argomento numerico viene convertito in una stringa esadecimale. Questa sarà lunga 2 caratteri se il numero com valore assoluto sarà tra +255 e - 255 , sarà invece di 4 caratteri se il valore sarà compreso tra questi e 65535. Valori superiori a questo provocano la comparsa del messaggio "B Integer out of range".

```
FN H$(32)= "20"
FN H$(255)= "FF"
FN H$ (-64)= "C0"
```

La capacità di usare numeri negativi dovrebbe risultare utile ai programmatori in L/M per calcolare salti relativi all'indietro. Per indirizzare la visualizzazione del contenuto di memoria in esadecimale si può fare:

```
100 INPUT indirizzo
110 PRINT FN H$ (indirizzo);"  ";FN H$(PEEK indirizzo)
120 LET indirizzo = indirizzo + 1: GO TO 110
```

Per specificare l'indirizzo di partenza in esadecimale sostituite la ligea 100 con:

```
100INPUT"Indirizzo di partenza?";A$:LET indirizzo=FN D(A$)
```

FN I(partenza, stringa A, stringa B)

Cerca nella stringa A dalla posizione "partenza", la stringa B(questa funzione in altri BASIC viene chiamata INSTRING). Se la stringa viene trogata, il risultato sarà la posizione nella stringa A del prino carattere della stringa B, altrimenti il risultato è zero. La stringa A può avere una lunghezza qualsiasi, mentre la stringa B non può essere più lunga di 256 caratteri oppure otterrete "Invalid argument". Se "partenza" è zero otterrete "Subscript wrong". Se la stringa B è + lunga della stringa A oppure"partenza" è più grande della lunghezza della stringa A oppure se entrambe le stringhe hanno lunghezza zero otterrete come risultato Ø.
E' possibile rimpiazzare alcuni dei caratteri nella stringa B con degli asterischi che significano che ricerca ciò che c'è in quella posizione non è significativo:

```
PRINT FN I(1,A$,"S* TH")
```

cercherà nella stringa A$ qualcosa come SMITH,SMATH,SLOTH ecc...

L'unica volta in cui l'asterisco fa parte dei caratteri da ricercare è quando [illegible] compare come primo carattere nella stringa da ricercare. La possibilità di specificare la posizione di partenza da cui cercare è utile quando vi aspettate di trovare la stringa che cercate più di una volta. Il seguente esempio cercherà "TEST" pella matrice A$ e segnalerà tutte le volte che è stato trovato sullo scherme:

```
100 DIM A$(1000)
110 FOR n= 1 TO RND x 10 + 3
120 LET pos= RNDx 995
130 LET A$ (pos TO pos+3) = "TEST"
140 NEXT n
150PRINT " La parola TEST è stata nascosta in A$ un numero
          random di volte, premi un tasto per scoprirle"
160 PAUSE Ø
170 LET loc=1
180 LET loc= FN I (loc,A$, "TEST")
190 IF loc<>Ø THEN PRINT "Obiettivo trovato alla posizione";
    loc: LET loc=loc+1: GO TO 180
```

L'array A$ comincia ad essere setacciato dalla posizione numero 1 poi con la linea 190 si segnala tramite scherme la locazione dove è stata trovata la parola "TEST" e quindi reinizia la ricerca dalla posizione immediatamente successiva. Quando FN I ="Ø" la ricerca significa che è stata completata.
Nota: la linea 190 potrebbe leggersi:

```
IF loc THEN PRINT....
```

perchè "Ø" significa false e quindi la successiva istruzione non verrebbe eseguita. Se pensate che l'esempio qui sopra rappresenti una situazione troppo semplice (dopo tutto la stringa è piena di spazi) cambiate la linea 100 con:

```
100 LET A$= FN S$(250,"TESS")
```

FN I può essere usato per controllare stringhe in INPUT in programmi educazionali o giochi. Per esempio supponete che un programma vi ponga una domanda la cui risposta dovrebbe essere "Napoleone" racchiuso in C$. A questo punto colui che risponde potrebbe introdurre la risposta in modo sintatticamente non esatto come " Napoleone"(notate lo spazion prima della parola) , oppure"Napoleone Bonaparte" e non riceverà altro che messaggi d'errore mentre la risposta è esatta credendo così di aver fornito la risposta sbagliata, ma questo succede solo se le due stringhe vengono comparate nel modo classico. Il problema può essere risolto con una linea di programma simile alla seguente che stampa"corretto"se nella stringa in INPUT è contenuta in qualche modo la risposta giusta:

```
INPUT A$:IF FN I(1,A$,C$)<>Ø THEN PRINT "Corretto"
```

Un'altra possibile applicazione potrebbe essere di implementare l'impaginettamento in una grande stringa di stringhe più piccole a lunghezza variabile.

Un modo potrebbe essere riservare tutti i caratteri in un certe range (ad esempio daCHR$ 1 aCHR$ 31) come marcatori di inizio e fine di ogni sottostringa. CHR$ 1 potrebbe indicare la prima sotto stringa, CHR$ 2 potrebbe indicare la seconda sottostringa e cosi via:

PRINT A$(FN I(1,A$,CHR$ n) + 1 TO FN I (1,A$, CHR$(n+1))-1)

stamperà la stringa numero "n" dalla stringa più grande A$. E' possibile usare molti altri schemi ma tutti hanno il vantaggio di una ricerca molto veloce della sottostringa richiesta e l'ulteriore vantaggio di risparmiare memoria se le stringhe sono a lunghezza variabile.

FN M()

FN M() fornisce il numero dei bytes liberi in memoria. Tra le parentesi non va messo niente. Provate:

PRINT FN M(): DIM A$(100): PRINT FN M()

Questa funzione molto semplice consiste solo di una chiamata alla ROM e può essere effettuata senza BetaBasic residente facendo:

PRINT 65535 - USR 7962

FN N (stringa)
Guardate pure: FN C$(numero)

Converte una stringa di 2 caratteri in un numero intero compreso tra 0 e 65535. L'equivalente in BASIC è:

LET numero= 256 x CODE C$(1) + CODE C$(2)

Otterrete "Invalid Argument" se la stringa non è lunga due caratteri: Con la funzione complementare FN C$, FNN rende facile l'implementazione di array interi con numeri interi.

FN P (indirizzo)
Guardate pure: DPOKE

DPOKE rappresenta un "doppio POKE" di uno specifico indirizzo e del byte successivo. L'equivalente Basic è

LET val= PEEK(indirizzo)+ 256xPEEK(indirizzo + 1)

Notate che il byte meno significativo è ilprimo ed è ciò che avviene sempre nel L/M e nelle variabili di sistema.
Per esempio:

```
100 LET NXT = FN P (23637): PRINT NXT: POKE NXT+5,65
110 REM XXXXX
```

Leggerà l'indirizzo della linea 110 dalla variabile di sistema NXTLN e il primo carattere dopo REM viene cambiato in "A" (i 5 bytes aggiunti la variabile NXT servono per superare il numero di linea e il numero

re

re dei bytes di linea).
DPOKE permette un doppio POKE nello stesso modo in cui FN P permette un doppio PEEK.

FN S$ (numero , stringa)

Questa funzione, in altri Basic, viene chiamata STRING$. Il suo risultato è il numero di ripetizioni della stringa.

```
FN S$(32,"-") = 32 segni"-"
FN S$(4, "AB")= "ABABABAB"
PRINT FN S$(704,"X")= uno schermo pieno di  "X"
PRINT FN S$(3, "A"+CHR$ 13)= A
                             A
                             A
```

FN S$ è più veloce di ciclo FOR-NEXT e più corto da digitare la stringa esplicitamente se più lunga di almeno 14 caratteri.

FN T$()
Guardate pure: CLOCK

Questa funzione ritorna l'ora corrente contenuta nell'orologio delBeta Basic. Se questo non è stato attivato FN T$()= "00:00:00". Una volta che l'orologio sia stato avviato sia che sia visualizzato cheno FN T$() fornira valori sempre varianti:

```
100 CLOCK 1
110 LET N$ = FN T$: PRINT N$
120 PRINT " ora=";N$(1TO 2); "min=";N$ (4 TO 5)
130 GO TO 110
```

Potrebbe essere una buona idea memorizzare il valore di FN T$ in un variabile così da poterlo usare, se necessario.

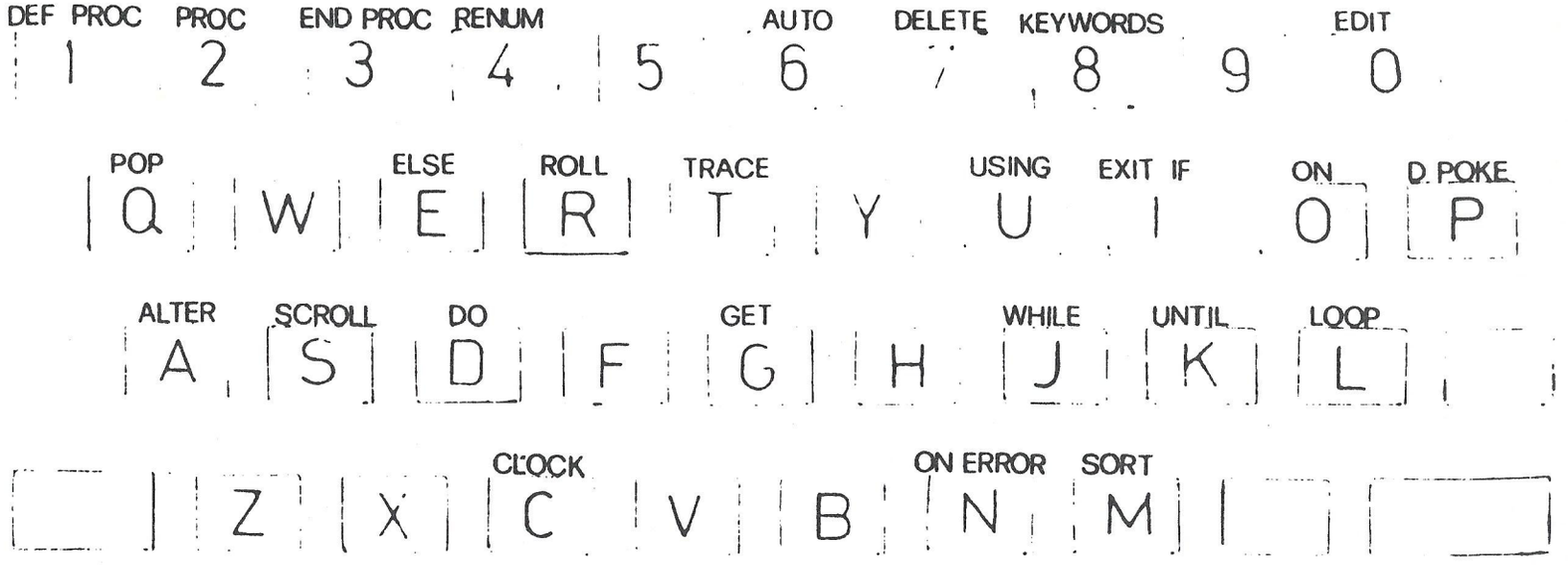
FN U$ (formato stringa,numero)
Guardate pure: USING

Restituisce la stringa equivalente del "numero" formattata seconda la "stringa formato". La differenza tra FN U$ e USING è che quest'ultimo può essere usato ~~xxx xxxxxxxx~~ comando solo con PRINT mentre il primo può essere usato con qualsiasi comando rigurdante le stringhe (es.:LET). Per ulteriori spiegazioni guardate USING.

MESSAGGI D'ERRORE

del BETA BASIC

CODICE	SIGNIFICATO	SITUAZIONE
B	"No room for line"	RENUM
	E' lo stesso messaggio del Sinclair ma ha un altro significato. Può comparire rinumerando il programma entro uno specificato blocco di linee e può accadere che il numero di linee presenti in quel blocco sia troppo alto per i numeri utilizzabili e si andrebbe quindi fuori blocco, oppure se fossenecessario usa numeri superiori a 9999	
S	"Missing LOOP"	DO, EXIT IF
	EXIT IF e un DO CONDidionale(seguito da WHILE o UNTIL) cercano di arrivare alla fine del loop per andare oltre ma non trovano LOOP	
T	"LOOP without DO"	LOOP
	Loop senza DO	
U	"No such line"	DELETE
	DELETE è stato usato con un numero di linea che non esiste nel programma.	
V	"No POP data"	POP
	E' stato effettuato un tentativo di rimuogere dati dallo stack che invece era vuoto(non c'era nessun GOSUB, DO-LOOP, PROC in azione)	
W	"Missing DEF PROC"	PROC,END PROC
	E' stato usato PROC senza che sia stata definita una procedura con quel nome op è stato usato END PROC senza DEF PROC.	
X	"No END PROC"	DEF PROC
	Cercando di saltare oltre una procedura non chiamata il programma non ha trovato END PROC.	
Y	"Too hard"	RENUM
	Effettuando una rinumerazione è stato trovato un riferimento a una linea sotto forma di espressione.	


DEF PROC
PROC
END PROC
RENUM
AUTO
DELETE
KEYWORDS
EDIT
1 2 3 4 5 6 7 8 9 0
POP
ELSE
ROLL
TRACE
USING
EXIT IF
ON
D.POKE
Q W E R T Y U I O P
ALTER
SCROLL
DO
GET
WHILE
UNTIL
LOOP
A S D F G H J K L
CLOCK
ON ERROR
SORT
Z X C V B N M